An Algorithm for Mining Frequent Itemsets from Library Big Data

Xingjian Li
lixingjianny@163.com
Library, Nanyang Institute of Technology, Nanyang, Henan 473000, China

***Abstract*—Frequent itemset mining plays an important part in college library data analysis. Because there are a lot of redundant data in library database, the mining process may generate intra-property frequent itemsets, and this hinders its efficiency significantly. To address this issue, we propose an improved FP-Growth algorithm we call RFP-Growth to avoid generating intra-property frequent itemsets, and to further boost its efficiency, implement its MapReduce version with additional prune strategy. The proposed algorithm was tested using both synthetic and real world library data, and the experimental results showed that the proposed algorithm outperformed existing algorithms.**

***Index Terms*—big data; frequent itemset; data mining; library**

I. Introduction

Frequent Itemsets (Frequent Patterns) Mining is an important technique in data mining, and it has been used in many fields, such as study of other data mining techniques, bank, library, etc. Since the first algorithm of mining frequent itemsets named Apriori was proposed by Rakesh Agrawal [1], new algorithms have been proposed constantly for various sub-domains of frequent itemsets mining, and these algorithms have been applied to many domains [2-5]. The algorithms Apriori and FP-Growth [1, 6] are representative of level-wise algorithms and pattern-growth algorithms, respectively; and many algorithms are variants of these two algorithms.

While many algorithms [7-13] have significantly increased their performance, they are still not fast enough for dealing with large datasets. This situation gave rise to the development of parallel algorithms. Along with the rapid growth of the application of MapReduce parallel computing framework, more and more data mining algorithms have been ported to run on this framework; parallel algorithms of mining frequent itemsets on MapReduce are based on either Apriori or FP-Growth. The algorithms in the papers [14-18] are based on Apriori, and they needed multiple MapReduce processes; if the length of the longest frequent pattern is *K*, *K* rounds of MapReduce must be applied. The algorithm PFP [19] is based on FP-Growth, and it needs only two rounds of MapReduce, but the data distributed to each node are heavily redundant, and the size of data chunks cannot be well-proportioned, so its time-performance is still not very satisfying.

Because there are a lot of redundant data in some datasets, such as library database, the mining process may generate intra-property frequent itemsets, and this hinders its efficiency significantly.

To address the above issue, we propose a parallel algorithm of mining frequent itemsets that needs at most only two rounds of MapReduce (in our experiments, only 1 round is needed in most cases), and this algorithm distributes data evenly among data nodes. Meanwhile, it can avoid generating intra-property frequent itemsets, and to further boost its efficiency. Experimental results confirmed that our algorithm outperformed existing algorithms significantly in term of time efficiency.

The rest of this paper is organized as follows: Section 2 gives the problem definitions; Section 3 proposes the new algorithm; Section 4 is the experiment results; and Section 5 is the conclusion.

II. Problem Definitions

Let a dataset $DB = \{t_1, t_2, \ldots, t_n\}$ contains n transaction itemsets and m distinct items $I = \{i_1, i_2, \ldots, i_m\}$. Each transaction $T = (TID, P)$ in DB consists of an itemset P and a unique identifier TID and $P \subseteq I$. $|DB|$ represents the size of dataset DB. An itemset $X = \{i_{j_1}, i_{j_2}, \cdots, i_{j_k}\} (1 \le j_1 < j_2 < \cdots < j_k \le m)$ containing k distinct items is called a k-itemset and k is the length of the itemset X.

Definition 1. The *minimum support threshold* *min_Sup* is the user specified percentile of number of transactions in the given dataset *DB*; then *minimum support number*, *min_SupNum*, in *DB* is defined by *min_SupNum* = *min_Sup* * |*DB*|.

Definition 2. The *support number* (*sn*) of an itemset *X* is the number of transaction itemsets containing *X*.

Definition 3. An itemset *X* is a frequent itemset if its support number is not less than the minimum support number.

Definition 4. Let dataset *DB* be divided into *s* little datasets $DB = DB_1 \bigcup DB_2 \bigcup \ldots \bigcup DB_s$. $min\_Sup * |DB_i|$ is called the local minimum support number of DB_i ($1 \le i \le s$). In dataset DB_i, an itemset *X* is a local frequent itemset if its support number is not less than the local minimum support number.

doi:10.4304/jsw.9.9.2361-2365

Property 1. Any subset X ($X \neq null$) of a frequent itemset is a frequent itemset; any superset of a non-frequent itemset is not a frequent itemset.

III. The Proposed Algorithm

This section mainly includes three parts: (1) preprocess the college library data; (2) give a new algorithm based on FP-Growth Algorithm; (3) implement the improved parallel FP-Growth Algorithm on MapReduce framework.

A. Preprocessing Library Data

This paper presents a data preprocessing method to process the combined data of books' bibliographic data, reader's information and readers' borrowing information by the following: (1) replacing all the property items of the book information with "A" prefix of consecutive numbers; (2) replacing all the property items of reader information with "B" prefix of consecutive numbers.

Definition 5. A frequent itemset is called intra-property frequent itemset if each item of the frequent itemset has the same prefix or property.

In most cases, the user doesn't care about the intra-property frequent itemsets, or association rules derived from intra-property frequent itemsets, but need to find out the frequent itemsets between different properties, such as discovering frequent itemsets or association rules between the book information and the reader information.

B. The Proposed Algorithm RFP-Growth

In many cases, the algorithm FP-Growth outperforms Apriori in terms of the mining efficiency. Therefore, this paper will improve the FP-Growth algorithm for mining data with a lot of redundant. The improved algorithm can avoid generating intra-property frequent itemsets, in order to improve the efficiency of mining. The improved algorithm is called RFP-Growth. Steps of RFP-Growth Algorithm are as follows:

Step1: Scan the dataset, count the support number of each item (the number of item appears in the dataset), then save the items and corresponding support number to a header table *H*. Each header table contains three fields: item name, support number and *link* field, and the field *link* records all nodes of an item on a tree.

Step2: Remove items whose support number is less than the minimum support number from header table. If the remaining items in this header table belong to one property or this header table is empty, then quit the mining algorithm. Otherwise, sort the remaining items by support number in descending order.

Step3: Scan the dataset for a second time to process the transaction item sets, the processing steps are as following steps:

Step3.1: Remove the non-frequent items from transaction itemsets.

Step3.2: Sort the remaining items of transaction itemsets by the order of items in header table, then adding the ordered itemsets to a tree *T*.

Step3.3: After ordered itemsets are added to a tree *T*, all the support number of the itemsets' corresponding nodes plus 1, then keep all the new nodes in the field *link* of corresponding header table.

Step4: Process each item of the header table from the last one. The processing steps are as following steps (assuming the current processing for item *Q*):

Step4.1: Add item *Q* into a base itemsets *BI*;

Step4.2: In header table *H*, *Q.link* contains all the nodes in the tree *T* whose item name is *Q*, denoted *k* nodes: *N*1, *N*2, ..., *Nk*.

Step4.3: read all the items from node *Ni*(*i* = 1, 2, ..., *k*) to the root of tree *T*, then save this items and its support number(the support numbers of each item are the same, which is the support number of node *Ni*) to a sub-header table *subH*(sub-header table subH has the same structure with the header table H).

Step4.4: Remove items from subH whose support number is less than minimum support number; if items of base itemsets BI and items of sub-header table subH are the same property or subH is empty, then step 5. Otherwise, sort the sub-header table subH by support number in descending order.

Step4.5: Read all the items from node *Ni* (*i* = 1, 2, ..., *k*) to the root of tree *T* in turn, remove local non-frequent items from the read items;

Step4.6: Sort the remaining itemsets by *sub-header* table order, then add the ordered itemsets to a new subtree *subT*.

Step4.7: When ordered itemsets are added to the subtree *subT*, all the support number of the itemsets' corresponding nodes plus *s* (*s* is the support number of node *Ni*), then keep all the new nodes in the link of corresponding subhead table.

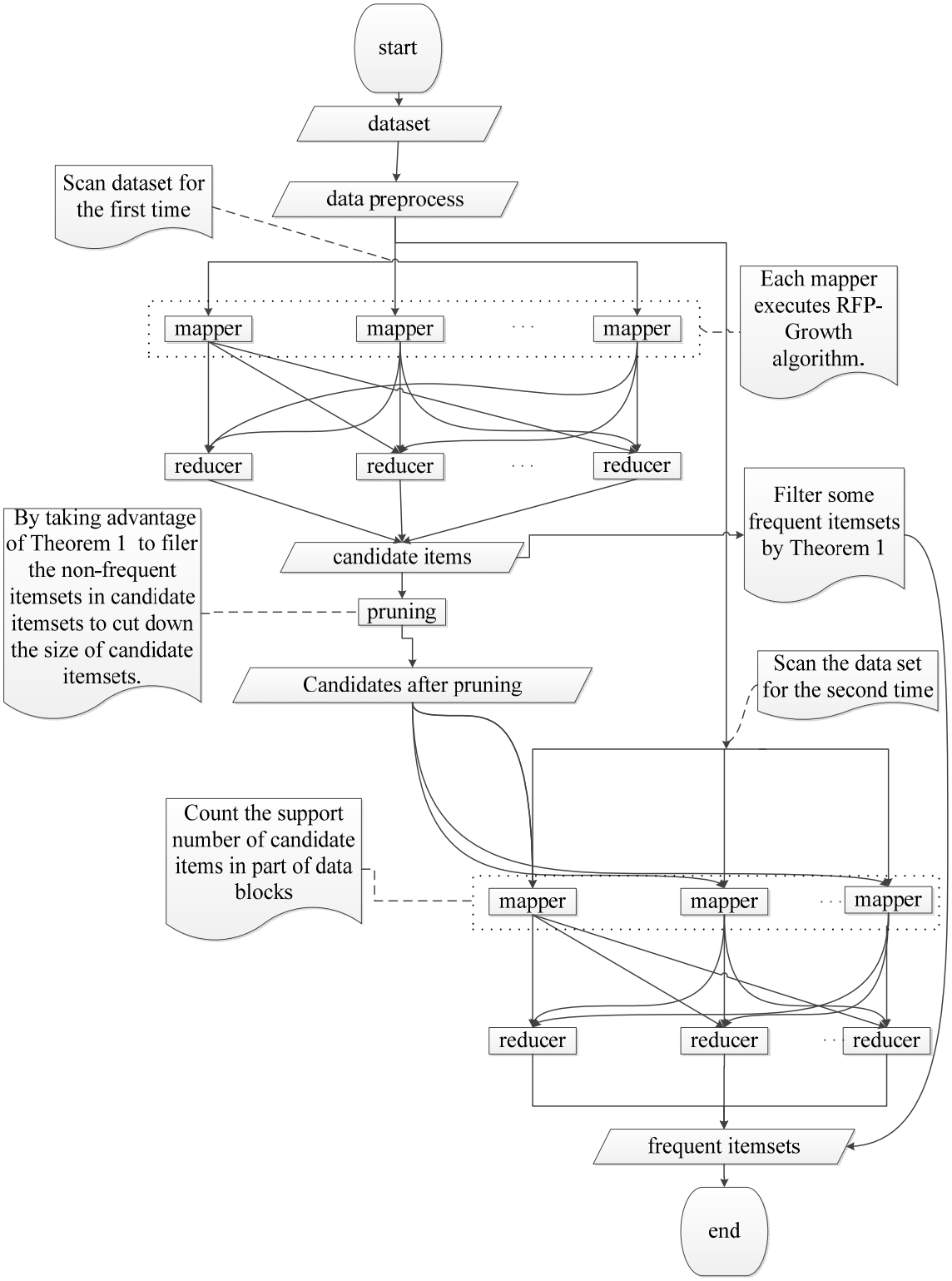


Figure 1. The flow chat of algorithm RFP-MR

Step4.8: Process sub-header table *subH* from step 4 recursively;

Step5: Remove the current processing item of the current processing header table from base itemsets BI, then continue process next item in the current processing header table.

The improved FP-Growth algorithm is called RFP-Growth, the main differences between the RFP-Growth algorithm and original FP-Growth algorithm are Step2 and Step4.4, and these two steps can effectively avoid generating intra-property frequent itemsets.

C. The Method of Mining Frequent Itemsets from a Library Big Data

Theorem 1: Itemset X is a frequent itemset if the sum of support number of part of the data blocks of the itemset X is not less than the minimum support number.

Proof: Obviously, if the sum of support number of part of the data blocks of an itemset X is not less than the minimum support number, then the sum of support number of all data blocks must have not less than the minimum support number. Therefore, the itemset X must be a frequent itemset.

Theorem 2: Divide dataset DB into m little datasets $DB = \{ DB_1, DB_2, \ldots, DB_m \}$, and let the frequent itemset contained in dataset DB be denoted by FI, and the local frequent itemset contained in each little dataset be called $FI_1, FI_2, \ldots, FI_m$. Then the frequent itemset of dataset DB is a subset of what on all the little datasets, that is $FI \subseteq FI_1 \cup FI_2 \cup \ldots \cup FI_m$.

Proof: Proof by contradiction.

Set the support number of a itemset X on the i-th data block as SN_i, then the support number of itemset X on dataset DB is $\sum_{i=1}^{m} SN_i$. If itemset X is not a frequent itemset on any data block, then the support number of X on any data block is less than the local minimum support number, that is $X_i < min\_Sup*|DB_i|$ $(1 \le i \le s)$. Therefore, $\sum_{i=1}^{m} X_i < \sum_{i=1}^{m} min\_Sup*|DB_i| = min\_Sup*\sum_{i=1}^{m}|DB_i| = min\_Sup*|DB|$. In conclusion, the itemset X is not a frequent itemset if it is not a local frequent itemset.

According to the principle of Theorem 2, mining frequent itemsets from a large dataset can be divided into two steps:

(1) Taking advantage of the algorithm RFP-Growth to mine local frequent itemsets from each data block, which generate candidate itemsets of global frequent itemsets.

(2) Re-scaning dataset to discover real frequent itemsets, where each step executes MapReduce once.

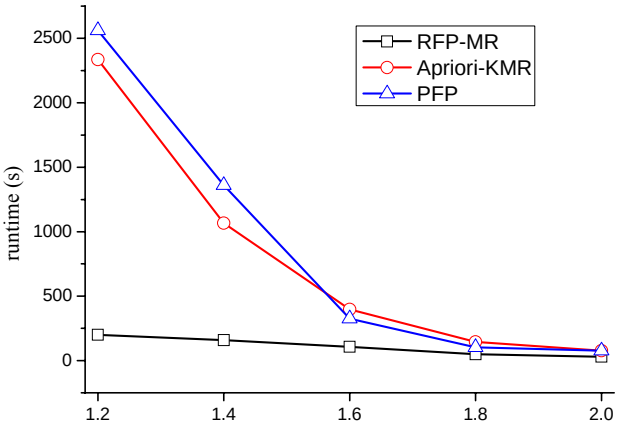
A parallel RFP-Growth algorithm based on MapReduce is called RFP-MR. Figure 1 is a flow chart of the algorithm RFP-MR. The transaction datasets execute MapReduce twice to mine frequent itemsets. Before performing the second round, cut down the size of candidate itemsets: remove some of the non-frequent itemsets from candidate itemsets by taking advantage of Property 1; discover some of the frequent itemsets by taking advantage of Theorem1. Thus, the time efficiency of the second MapReduce can be efficiently improved by decreasing of the size of candidate itemsets.

IV. EXPERIMENTAL RESULTS

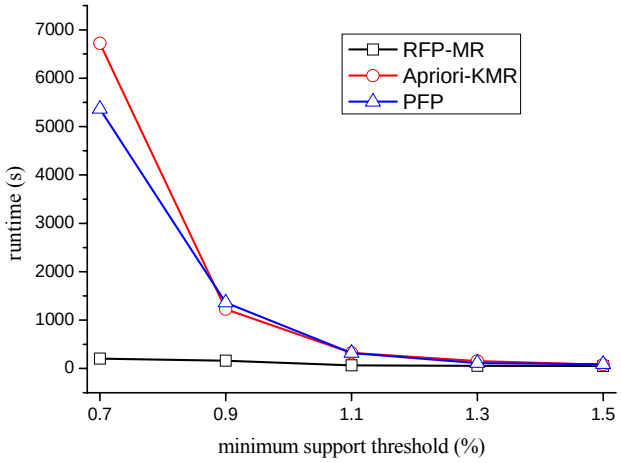
In order to verify the effectiveness of our proposed algorithm, we compare our algorithm with Apriori-based k rounds MapReduce (referred to as Apriori_KMR) algorithm [18] and the algorithm PFP [19] respectively. All experiments in this paper are based on the algorithm implemented in Python. Experimental platform is a cluster of 26 nodes, containing a master node, a scheduling node, a backup node and 23 data nodes. Each node in the hardware is configured with the 2.5GHz dual-core CPU and 8GB of memory, and the software is configured with ubuntu 12.04 and Hadoop 0.23.0.

In this paper, we take two datasets, one is historical borrowing data of our university (denoted as DATA1), it contains a total of 2000K transactions, and the length of each transaction itemset is 26 (that is each transaction contains 26 properties). The other one is synthetic dataset T20I10D10000K, and it was generated by IBM data generator, this dataset contains 10000K transactions，the average length of the transactions is 20. In order to fully taking advantage of 23 data nodes, the tested data file in this paper has been divided evenly into 20 small data files.

Figure 2 shows the runtime of three algorithms in different minimum support threshold, and it can be seen from Figure 2 that the proposed algorithm in this paper outperformed the algorithm Apriori_KMR in term of time efficiency, because with the decrease of minimum support threshold, the number of frequent itemsets increases. Thus, the algorithm Apriori-KMR produces excessive candidate itemsets, and it repeatedly invokes MapReduce. The proposed algorithm in this paper executes MapReduce for the first time to produce local frequent itemsets, and then cut down the size of candidate itemsets by filtering the candidate itemsets. In many cases, there is no need for a second round of MapReduce to count the support number of candidate itemsets. Therefore, the time efficiency of the proposed algorithm is higher than the algorithm Apriori_KMR. When testing data in DATA1, since the dataset has intra-property frequent itemsets, the algorithm Apriori_KMR can avoid generating intra-property frequent itemsets effectively. Therefore, the algorithm RFP-MR outperforms Apriori-KMR in term of time efficiency.



(a) The dataset DATA1



(b) The dataset T20I10D10000K

Figure 2. The runtime of three algorithms under varied minimum support threshold

In the algorithm PFP, the data distributed to each node are heavily redundant, and the size of data chunks cannot be well-proportioned, so its time efficient is still not very satisfying. As shown in Figure 2, the proposed algorithm RFP-MR outperforms PFP in term of time performance.

V. Conclusion

With the development of information technology, library data is also growing, in order to effectively mine the frequent itemsets, this paper have done three works, one is giving a mining algorithm RFP-Growth, which can effectively avoid generating intra-property frequent itemsets. Secondly, applying the improved algorithm RFP-Growth to the MapReduce framework, and realize parallelism, which can effectively mine frequent itemsets from big data. Finally, we use the library borrowing data and a synthetic dataset to validate the proposed algorithm RFP-MR, and the experimental results confirmed that our algorithm RFP-MR outperformed existing algorithms Apriori-KMR and PFP significantly in term of time efficiency.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. International Conference on Very Large Data Bases (VLDB 1994). 1994. Santiago, Chile.

[2] B. Li. Non-personalized Book Recommendation Based on Search Behavior. Library Journal, 2013. 8(32): pp. 36-41.

[3] Q. Ruan, H. Lin and H. Tan. Research on the approach to optimize book exhibition based on mining association rules. International Journal of Advancements in Computing Technology, 2012. 4(16): pp. 500-507.

[4] H. He. Analysis ofAssociation Rules in Book Circulation. Library Journal, 2011. 7(30): pp. 63-68.

[5] M. Antonie, O.R. Zaiane and A. Coman. Application of Data Mining Techniques for Medical Image Classification. MDM/KDD, 2001. 2001: pp. 94-101.

[6] J. Han, J. Pei and Y. Yin. Mining frequent patterns without candidate generation. ACM SIGMOD International Conference on Management of Data. 2000. Dallas, TX, United states.

[7] L. Wang, L. Feng and M. Wu. AT-Mine: An Efficient Algorithm of Frequent Itemset Mining on Uncertain Dataset. Journal of Computers, 2013. 8(6): pp. 1417-1426.

[8] L. Feng, L. Wang and B. Jin. UT-Tree: Efficient mining of high utility itemsets from data streams. Intelligent Data Analysis, 2013. 17(4): pp. 585-602.

[9] B. Chandra and S. Bhaskar. A novel approach for finding frequent itemsets in data stream. International Journal of Intelligent Systems, 2013. 28(3): pp. 217-241.

[10] J.J. Cameron, A. Cuzzocrea and C.K. Leung. Stream mining of frequent sets with limited memory. 28th Annual ACM Symposium on Applied Computing (SAC 2013). 2013. Coimbra, Portugal.

[11] B. Vo, T. Hong and B. Le. DBV-Miner: A Dynamic Bit-Vector approach for fast mining frequent closed itemsets. Expert Systems with Applications, 2012. 39(8): pp. 7196-7206.

[12] W. Shui and W. Le. An implementation of FP-growth algorithm based on high level data structures of weka-JUNG framework. Journal of Convergence Information Technology, 2010. 5(9): pp. 287-294.

[13] M. El-hajj and O.R. Zaïane. COFI-tree mining: a new approach to pattern growth with reduced candidacy generation. IEEE International Conference on Frequent Itemset Mining Implementations. 2003.

[14] T. Xiao, C. Yuan and Y. Huang. PSON: A parallelized SON algorithm with MapReduce for mining frequent sets. 4th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP 2011). 2011. Tianjin, China.

[15] M. Riondato, et al. PARMA: A parallel randomized algorithm for approximate association rules mining in MapReduce. 21st ACM International Conference on Information and Knowledge Management (CIKM 2012). 2012. Maui, HI, United states.

[16] X.Y. Yang, Z. Liu and Y. Fu. MapReduce as a programming model for association rules algorithm on Hadoop. 3rd International Conference on Information Sciences and Interaction Sciences (ICIS 2010). 2010. Chengdu, China.

[17] J. Cryans, S. Ratte and R. Champagne. Adaptation of apriori to MapReduce to build a warehouse of relations between named entities across the web. 2nd International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2010). 2010. Menuires, France.

[18] M. Lin, P. Lee and S. Hsueh. Apriori-based frequent itemset mining algorithms on MapReduce. 6th International Conference on Ubiquitous Information Management and Communication. 2012.

[19] H. Li, et al. PFP: Parallel FP-growth for query recommendation. 2nd ACM International Conference on Recommender Systems (RecSys 2008). 2008. Lausanne, Switzerland.